4) Subroutines

Matt Webster
IMBIM, BMC
matthew.webster@imbim.uu.se

Reminder: scalars

- variables in perl (e.g. `$example`) can be numbers or strings
- perl converts variable between the two depending on the operation
- backslash `\` used to indicate special characters like newline `\n` and tab `\t`
- compare numbers or strings with comparison operators and `if`
- `<STDIN>` gets input from the keyboard
- `chomp` removes `\n`
- `while` repeats a chunk of code while an expression is true

Reminder: arrays

`$array[0]`
- first element of `@array`

`$#array`
- is the final index of `@array`

`qw//`
- quoted by whitespace

`push pop`
- add or remove elements from the end

`shift unshift`
- add or remove elements from the start

`splice`
- add, remove, replace elements from the middle

`foreach`
- cycle through every element

`sort reverse`
- change order

`<STDIN>` can also be read into an array

subroutines

- We have already looked at some functions such as

```
chomp sort print
```

- One way to program in perl is to write subroutines, which are user-defined functions
- You call a subroutine using

```
&example;
```

Or

```
example();
```

- Arguments can go in the parentheses.

subroutines

```
@students = qw/Doreen Oskar Elin Sangeet Malin/;
```

- this is how to define a subroutine
- usually at the end of the script

```
sub next_student {
    $i++;
    print "the next student is $student[$i]\n";
}
```

```
&next_student
the next student is Oskar

&next_student
the next student is Elin

&next_student
the next student is Sangeet
```

return values

- subroutines have a return value
- the return value is the last thing evaluated by the subroutine

```
sub next_student {
    $i++;
    $student[$i];
}
```

```
$person = &next_student;
print "I found $person\n";
I found Oskar

$person = &next_student;
print "I found $person\n";
I found Elin
```

return values

- subroutines have a return value
- the return value is the last thing evaluated by the subroutine

```
@students = qw/Doreen Oskar Elin Sangeet Malin/;
$i=2;
$person = &first_student;
print "$person\n";

sub first_student {
    if ($student[$i] lt $student[$i+1]) {
        $student[$i];
    } else {
        $student[$i+1];
    }
}
```

```
Elin
```

- Script evaluates difference between `Elin` and `Sangeet`
- `Elin` is less than `Sangeet` (alphabetically)

arguments

- you can give arguments to a subroutine
- arguments are stored in special array called `@_`
- `@_` can be accessed with `$_[0]` `$_[1]` etc. etc.
- or you can use:

```
$first_arg = shift(@_);
```

```
$person = &first_student("Elin","Sangeet");
print "$person\n";

sub first_student {
    if ($_[0] lt $_[1]) {
        $_[0];
    } else {
        $_[1];
    }
}
```

```
Elin
```

my

- using `my` makes a variable private to a block of code

```
$person = &first_student("Elin","Doreen");
print "$person\n";

sub first_student {
    my $person = shift (@_);
    my $another_person = shift (@_);
    if ($person lt $another_person) {
        $person;
    } else {
        $another_person;
    }
}
```

```
Doreen
```

`my $person` is confined to the subroutine (not accessible outside)

my

- `my` is used all the time to define private variables

```
my $student;
my ($student,$teacher) = ("Sangeet","Matt");
my @students = ("Sangeet","Elin")

foreach my $student (@students) {
    print "the next student is $student\n";
}
```

```
the next student is Sangeet
the next student is Elin
```

```
print "$student\n"
# what does this give?
```

```
Sangeet
# because my restricts to the foreach
```

use strict

- this makes perl much more intolerant of your code
- useful if you want to avoid missing errors
- means you need to be more careful in your coding
- very helpful when debugging long programs with many loops and subroutines
- every variable must be declared with `my` the first time

```
$number_students = 22;
# not allowed with use strict

my $number_students = 22;
$number_students++;
#allowed
```

return

- use `return` to stop a subroutine and go back
- you can also use it to return a value

```
my @students = qw/Doreen Oskar Elin Sangeet Malin/;
my $result = &which_student_is("Elin", @students);

print "$result\n";

sub which_student_is {
   my($what, @array) = @_;
   foreach (0..$#array) {
      if ($what eq $array[$_]) {
         return $_;
      }
   }
}
```

2

other subroutine facts:

the `&` is usually not needed

```
&which_student_is("Elin", @students);
which_student_is("Elin", @students);
#these are the same – it is obviously a subroutine

&chomp
chomp
#the first is a subroutine, the second is a standard perl function
```

the return value can be an array

```
return (1..5)
#will give (1,2,3,4,5)
```

summary

- subroutines
  - called by `&sub_example` or `sub_example()`
  - defined using `sub { }`
  - use `return` or it will return last evaluated value
  - arguments passed within:

  `@_`

  `$_[0], $_[1], $_[2]`….

- `my` is used to restrict a variable to a block of code
- `use strict` means all variables must be declared using `my`